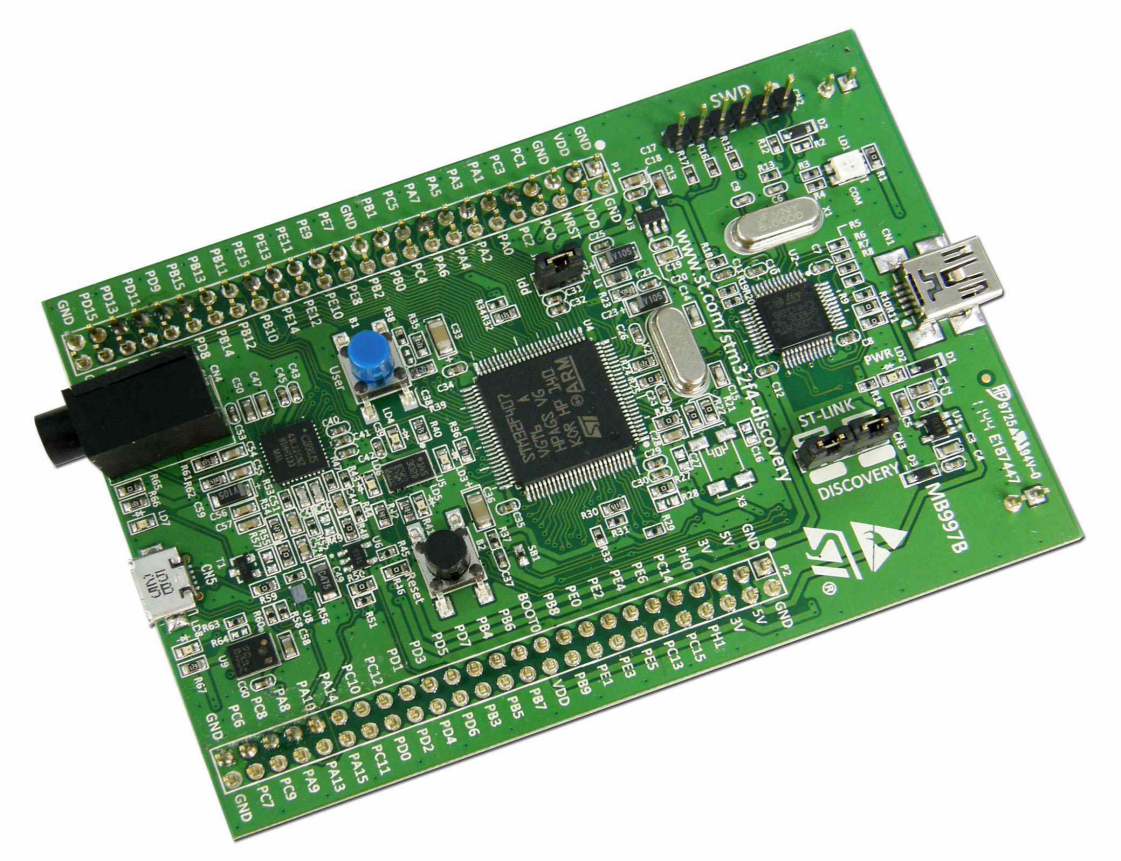
Podstawy technik mikroprocesorowych
Laboratorium

Obsługa timerów mikroprocesora SMT32F4VGT6



Politechnik Poznańska
Wydział Elektryczny
Instytut Automatyki i Inżynierii Informatycznej

Adam Owczarkowski
Poznań 18.10.13

1. Wprowadzenie

Omawiany mikrokontroler posiada czternaście sprzętowych timerów. Mają one wiele zastosowań, choć głównie są one powiązane z generacją zdarzeń w ściśle określonych chwilach czasowych niezależnie od wykonywanego programu przez procesor. Do timerów przypisane jest kilka rejestrów konfiguracyjnych, jednak najważniejszym jest licznik impulsów. Ich wartości mogą być zwiększane albo przez sam procesor albo przez przebiegi cyfrowe pochodzące ze świata zewnętrznego.

2. Konfiguracja

- Konfiguracja dystrybucji czasu w procesorze.
  Należy podmienić plik system_stm32f4.c z podanym przez prowadzącego. Następnie jednokrotnie wywołać dwie funkcje:

```
SystemInit();
SystemCoreClockUpdate();
```

- Dołączenie pliku konfigurującego timery.
  Należy dodać w zakładce Repository plik opisany TIM oraz dodać w kodzie programu plik nagłówkowy

```
#include "stm32f4xx_tim.h"
```

- Podłączenie timera do magistrali.

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIMx, ENABLE);
```

  gdzie x to numer timera (**należy zamiast x wpisać numer timera!**).

- Funkcja inicjująca podstawę czasu timera.

```
TIM_TimeBaseInit(TIMx, &TIM_TimeBaseStructure);
```

  gdzie `TIM_TimeBaseStructure` to struktura, której pola należy wypełnić odpowiednimi danymi. Inicjalizacja struktury to:

```
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
```

  Kolejne pola struktury oznaczają:
  - `TIM_Prescaler` – dzielnik licznika,
  - `TIM_CounterMode` – tryb zliczania licznika:
    - `TIM_CounterMode_Up` – w górę,
    - `TIM_CounterMode_Down` – w dół,

- `TIM_Period` – okres licznika,
- `TIM_ClockDivision` – dzielnik zegara (należy ustawić `TIM_CKD_DIV1`).

Gotowy fragment kodu to:

```
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIMx, ENABLE);

/* Time base configuration */
TIM_TimeBaseStructure.TIM_Period = ...;
TIM_TimeBaseStructure.TIM_Prescaler = ...;
TIM_TimeBaseStructure.TIM_ClockDivision = ...;
TIM_TimeBaseStructure.TIM_CounterMode = ...;

TIM_TimeBaseInit(TIMx, &TIM_TimeBaseStructure);
```

W miejscu kropek należy wpisać odpowiednie wartości.
Wzór matematyczny określający częstotliwość inkrementacji/dekrementacji licznika:

$$f = \frac{f_{cpu} \cdot period}{2 \cdot prescaler}$$

gdzie f_{cpu} to częstotliwość taktowania procesora (w rozważanym przypadku to 168 MHz).

- Funkcja uruchamiająca timer:

```
TIM_Cmd(TIMx, ENABLE);
```

- Sczytywanie aktualnej wartości licznika.

```
unsigned int counter;
counter=TIMx->CNT;
```

- Dołączenie pliku konfigurującego przerwania.
Należy dodać w zakładce Repository pliki opisany EXTI oraz MISC oraz dodać w kodzie programu pliki nagłówkowe

```
#include "stm32f4xx_tim.h"
#include "misc.h"
```

- Funkcja konfigurująca przerwania.

```
NVIC_Init(&NVIC_InitStructure);
```

gdzie `NVIC_InitStructure` to struktura, której pola należy wypełnić odpowiednimi danymi. Inicjalizacja struktury to:

```
NVIC_InitTypeDef NVIC_InitStructure;
```

Kolejne pola struktury oznaczają:

- `NVIC_IRQChannel` – określa co ma generować przerwanie (w tym przypadku *`TIMx_IRQn`*),
- `NVIC_IRQChannelPreemptionPriority, NVIC_IRQChannelPreemptionPriority` określają priorytety przerwania (należy wpisać wartość 0),
- `NVIC_IRQChannelCmd` – uruchamianie/wyłączanie kanału przerwań (należy wpisać *`ENABLE`*).

Gotowy fragment kodu to:

```
NVIC_InitTypeDef NVIC_InitStructure;
NVIC_InitStructure.NVIC_IRQChannel = ...;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = ...;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = ...;
NVIC_InitStructure.NVIC_IRQChannelCmd = ...;
NVIC_Init(&NVIC_InitStructure);

TIM_ClearITPendingBit(TIMx, TIM_IT_Update);
TIM_ITConfig(TIMx, TIM_IT_Update, ENABLE);
```

Dwie ostatnie linijki to odpowiednio zerowanie flagi przerwania timera i uruchomienie przerwania.

- Obsługa przerwania.

```
void TIMx_IRQHandler(void)
{
        if(TIM_GetITStatus(TIMx, TIM_IT_Update) != RESET)
        {
        // miejsce na kod wywoływany w momencie wystżpienia przerwania
              TIM_ClearITPendingBit(TIMx, TIM_IT_Update);
        }
 }
```

Powyższą funkcję należy umieścić w kodzie programu i wypełnić. Dana funkcja będzie wywoływana za każdym wystąpieniem przerwania.

- Konfiguracja wyjścia PWM.

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
```

Przy podstawowej konfiguracji wejść/wyjść należy dany pin ustawić w tryb funkcji alternatywnej jak wyżej pokazano. Niezbędne jest również „przemapowanie” pinu pisząc:

```
GPIO_PinAFConfig(GPIOD, GPIO_PinSource12, GPIO_AF_TIM4);
```

Powyższa funkcja sprawia, że wyjście PORTD12 jest podłączone do timera 4.

- Konfiguracja timera w tryb PWM.

```
TIM_OCInitTypeDef  TIM_OCInitStructure;
/* PWM1 Mode configuration:  */
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = 0;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
```

Powyższy kod odpowiada za konfigurację timera w trybie generowania przebiegów cyfrowych z modulacją szerokości impulsu PWM (Pulse Width Modulation). Struktura `TIM_OCInitStructure` daje możliwość zaawansowanego ustawiania różnych parametrów przebiegu. Teraz jeszcze należy podłączyć kanał OC1 timera:

```
TIM_OC1Init(TIMx, &TIM_OCInitStructure);
TIM_OC1PreloadConfig(TIMx, TIM_OCPreload_Enable);
```

Oczywiście nie można zapomnieć o właściwej konfiguracji podstawy czasu timera, aby przebieg był o właściwej częstotliwości.

- Ustawienia poziomu wypełnienia przebiegu PWM.

  Wypełnienie sygnału ustawia się zgodnie z poniższym wzorem:

  $$duty = \frac{CCR}{period} \cdot 100\,[\%]$$

  gdzie CCR to Capture/Compare Register (rejestr porównywania), a period to okres wcześniej ustawiony w timerze. Ilekroć pojawia się potrzeba zmiany wypełnienia sygnału należy aktualizować zawartość rejestru CCR tak jak poniżej:

```
TIM4->CCR1 = 10; // set brightness
```

  Jeden timer może generować wiele przebiegów PWM przy pomocy wielu kanałów i dlatego posiada on wiele rejestrów CCR, np. CCR1, CCR2 itd. Powyżej przedstawiono sposób wpisywania liczby do rejestru CCR1 timera 4.

3. Zadania

- Napisać program, który powoduje mruganie diody LED przy pomocy timera. Niech to będzie timer nr 4, a dioda podłączona do wyjścia na PORTD12. Skonfigurować właściwie częstotliwość i okres timera. W pętli głównej programu regularnie sprawdzać licznik timera i niech dioda LED się zapala po przekroczeniu odpowiedniej liczby. Celem jest, aby dioda przez jedną sekundę się świeciła i jedną sekundę była wyłączona.
- Zrealizować ten sam efekt jak w powyższym zadaniu, lecz korzystając z

przerwań timera nr 4.

- Napisać program, który powoduje świecenie diody LED z różnym natężeniem światła korzystając z modulacji szerokości impulsu PWM. Nadal nie będzie to dioda LED podłączona do portu PORTD12 oraz timer o numerze 4. Celem zadania jest wygenerowanie przebiegu o częstotliwości 100 Hz o wypełnieniach zmieniających się co około jedną sekundę pomiędzy wartościami: 0 %, 10 %, 50 % i 100 %. Przebieg ten ma obrazować dioda LED.